Comp3100 Assignment 1 Report

Cloud Job Scheduler

Huu Hoang Khoi Dao 45253927
Aashir Hussain 45269114
Ngere Papera 44901445

- Introduction

Our project aims to develop a job scheduler for distributed systems. The job scheduler is designed to be used for cloud data centres. It allows you to schedule essentially any job, including retail, data jobs, and more. The system is automated, reducing manual inputs allowing for quick and efficient dispatching of jobs. The program currently acts as a simple job dispatcher that takes the selected jobs and places them into the desired queue.

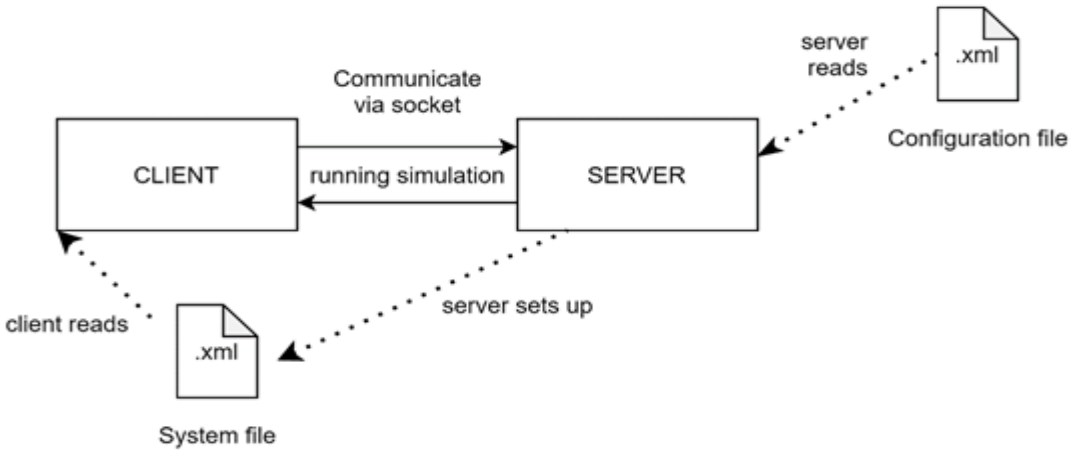
- System Overview

This system consists of four main components:

- Client
- Server
- Configuration XML file
- System XML file

The aim of this system is to establish a 3 way handshake between the client and the server. This is then followed by the client reading the system.xml file to retrieve the status of available servers.

Figure: 1



Communication between the client and the server is conducted using commands such as “HELO”, “OK”, “REDY”.

1. The client sends “HELO” to the server and awaits a response.
2. Server responds with “OK”.
3. The client then sends “AUTH” with the username to the server.
4. Server responds with “OK”.
5. Server will have then printed a welcome message and written to system info.
6. The client then sends “REDY” to the server.

After this, jobs will be dispatched and carried out via client-server communication. Users will submit jobs and be queued for assignment. The job scheduler will assign these jobs depending on server type as these servers have different specifications and varying statuses.

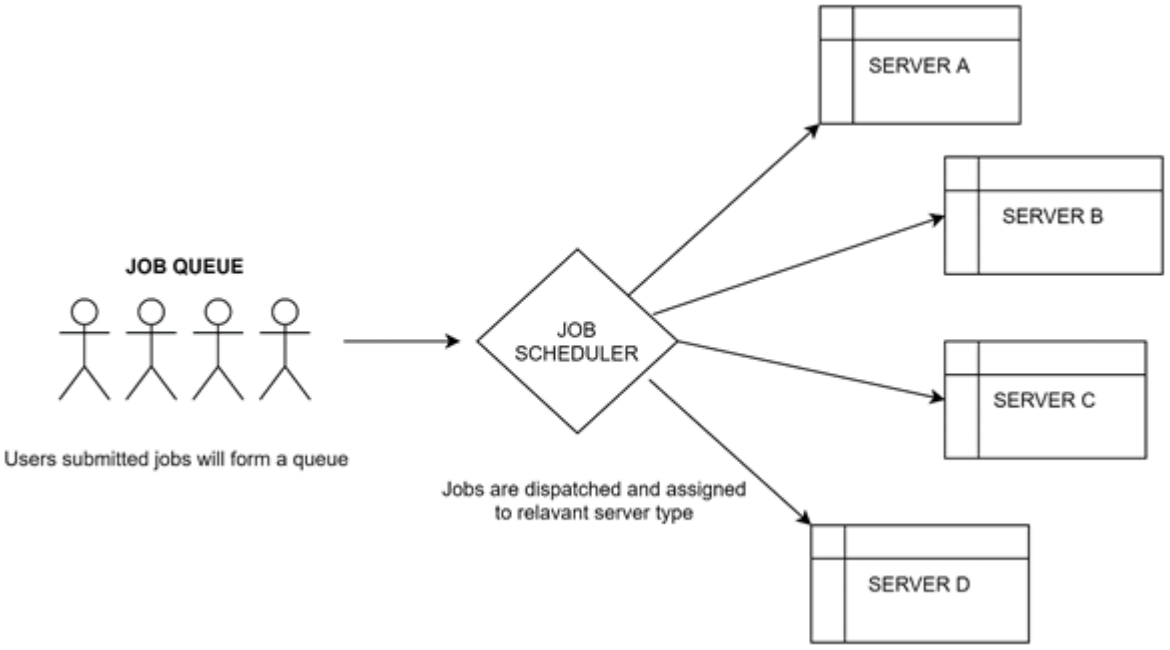
A job will contain identifiers such as

- Timing (i.e.; submission time)

- Resource requirements (memory, CPU cores, size MB)

These differences in specifications require the jobs to be assigned to the suitable server. The cloud job scheduler will first determine the largest server by searching through the list of servers and determining based on the highest individual core count.

Figure: 2



- Design

Design philosophy

What we wish to accomplish with this Scheduling System is to make a distributed system that omits everything superfluous and unnecessary, to show only the essential for a quick and efficient cloud-based job scheduler. We also wanted to focus on flexibility, adaptability, and scalability so it can run on virtually any operating system architecture. Therefore, we have chosen Java as the developmental language. Java by design is made to be independent, that allows for ports to different architectures easier.

Consideration and Constraints

It is very difficult for our group to provide security in the distributed system as it requires both the nodes and connections to be secured.

Some messages and data can be lost in the network while moving from one node to another.

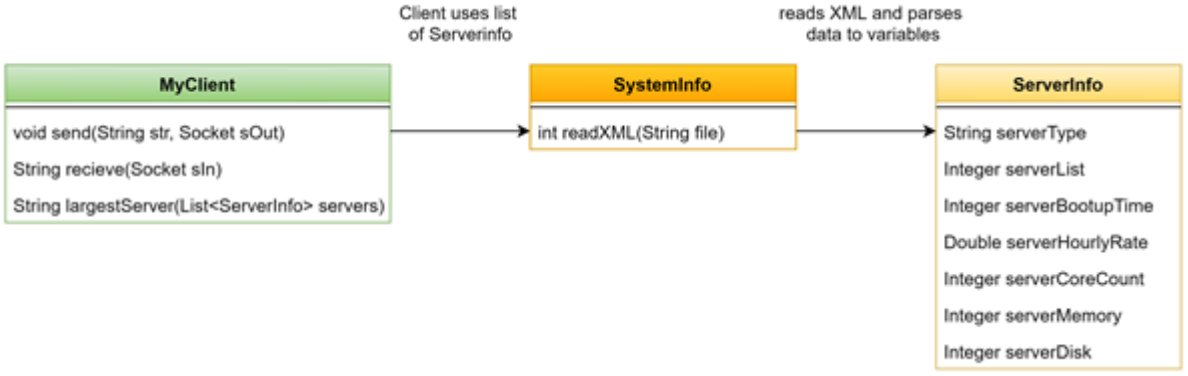
Overloading may occur in the network if all the nodes of the distributed system try to send data at once.

We chose to use XML because it is easier to reference each attribute of each server.

We chose to use DataInputStream over BufferedInputStream because it is better for reading primitives and length-prefixed strings.

Functionalities of each simulator component

Figure 3:



- Implementation

The Cloud Job Scheduler implementation consists of 3 classes.

MyClient.java, ServerInfo.java and SystemInfo.java

Within the MyClient class there are 5 methods that are declared and used to perform certain actions required to connect and talk to the server, as well as, notify any errors that occur during the connection.

Main - In the main method we create a new object called “Socket”. The socket allows for peer-to-peer communication between the client and server. The socket interfaces with the network protocol TCP/IP. The loop in the main method checks for command line arguments. There is also a handshake that allows the client that authenticates itself to the server.

Send - This method allows messages to be sent from the client side. It utilises DataOutputStream to send the appropriate bytes of the specified string.

Receive - This method allows messages to be received from the server side. It utilises DataInputStream to read the bytes of each command sent by the server.

LargestServer - This method searches through a list of servers that have been read from the ds-system.xml file and determines the largest server based on the highest individual core count. If there are 2 servers of the same size, the first will be selected.

Error - This method handles errors and ensures that the client safely exits the simulation.

ReadXML - The method is within the SystemInfo class. This method uses data parsing to read xml and converts to appropriate data types (Strings, Integers and Doubles). It determines each server’s unique attribute and stores it as a ServerInfo object.

Libraries used:

import javax.xml.stream.XMLInputFactory - Implementation of a factory for getting streams. This aids in the reading of xml files.

import javax.xml.stream.XMLStreamConstants - This interface declares the constants used with the API XML.

import javax.xml.stream.XMLStreamException - This is used to check for unexpected processing errors.

import javax.xml.stream.XMLStreamReader - Iterates over XML format by using next() and hasNext(). This allows read-only access to XML at the lowest level.

Tasks were divided equally between all 3 members. Each had a hand in both the report and the job dispatcher code. Neri led most of the development for the dispatcher code, while Huu and Ash handled the majority of the documentation.
The github log and google docs edit history will show that work was done periodically by each member.

- References

https://github.com/hoangdao46/Scheduling-in-Distributed-Systems.git

https://docs.google.com/document/d/14jXd-GwEuZrtCjJMODx_CqnmeN4pc5LHIOqmOxZT8ME/edit?usp=sharing